Kontinuirano testiranje i metrike u DevOps ekosistemu

Kako povezati programersko okruženje sa produkcijom i osigurati kvalitet u brzom ciklusu isporuke

Agenda i ciljevi predavanja

01

Razumevanje problema

Zašto tradicionalno testiranje ne funkcioniše u DevOps-u.

02

Shift-Left i Shift-Right

Strategije ranijeg i kasnijeg pomeranja testiranja u životnom ciklusu.

03

Kapije kvaliteta

Automatizacija odluka u toku za kontinuiranu integraciju i isporuku (CI/CD pajplajnu).

04

Optimizacija testova

Kako izbeći sate čekanja na rezultate testova.

05

Praktične demonstracije

Live-coding kroz Javu, JUnit i Javalin.

Problem koji rešavamo: jaz između razvoja i produkcije

Tradicionalni ciklus razvoja

Programeri pišu kod, QA testira nedeljama kasnije, produkcija pati.

Brzi ciklus isporuke

U DevOps-u se kod isporučuje više puta dnevno.

Izazov

Kako osigurati da testovi imaju smisla i drže korak sa brzinom isporuke?

REŠENJE

Kontinuirano testiranje (Continuous Testing) kao integralni deo CI/CD pajplajna.

Šta je Continuous Testing (CT)?

Definicija: Proces izvršavanja automatizovanih testova kao dela pajplajna isporuke softvera.

Kontinuirana povratna informacija: Dobijanje informacija o poslovnim rizicima u najkraćem mogućem roku.

U SVAKOJ FAZI

Od koda do produkcije.

POTPUNO AUTOMATIZOVANO

Automatizovano izvršavanje.

STABILNO I PONOVLJIVO

Isti uslovi i ponovljiv rezultat.

Shift-Left vs. Shift-Right strategije

VREMENSKA LINIJA RAZVOJA



Shift-Left

Pomeranje testiranja što ranije u razvojni ciklus.

Shift-Right

Pomeranje testiranja u samu produkciju i realno okruženje.

Sinergija: Nijedan pristup nije dovoljan sam za sebe; oba čine kompletan CT.

Tradicionalno se testira na sredini/kraju životnog ciklusa.

Shift-Left strategija: teorijski koncept

Suština

Pronađi bag tamo gde je nastao – na programerskom stolu.

Cena greške: Što se greška kasnije otkrije, njeno otklanjanje je eksponencijalno skuplje.

KLJUČNE TEHNIKE

Statička analiza koda

Provera izvornog koda bez izvršavanja.

Test-Driven Development (TDD)

Pisanje testova pre produkcionog koda.

Pre-merge validacija

Pull Request provere pre spajanja.

Alat za Shift-Left: statička analiza koda

Definicija

Analiza izvornog koda bez njegovog izvršavanja.

Šta detektuje

Kršenje stilskih pravila, potencijalne bagove, bezbednosne propuste (SAST).

Java alati

Checkstyle, SpotBugs, PMD, SonarQube.

ULOGA U CI

Automatsko odbijanje koda koji ne ispunjava standarde pre nego što uopšte krene kompajliranje testova.

Praktičan primer: konfiguracija SpotBugs u Maven-u

Cilj: Automatizovati statičku analizu tokom Maven build-a. Kreiranje pom.xml dodatka:

POM.XML / MAVEN PLUGIN

```
<plugin>
    <groupId>com.github.spotbugs</groupId>
    <artifactId>spotbugs-maven-plugin</artifactId>
    <version>4.8.2.0</version>
    <executions>
        <execution>
            <phase>verify</phase>
            <goals><goal>check</goal></goals>
        </execution>
    </executions>
</plugin>
```

Loš kod koji SpotBugs detektuje (Java primer)

Primer baga (Null Pointer i poređenje stringova):

JAVA / USER SERVICE

```
public class UserService {
    public boolean checkAdmin(String role) {
        // Potencijalni NullPointerException ako je role null
        // i pogrešno poređenje referenci umesto .equals()
        return role == "ADMIN";
    }
}
```

KOMANDA `mvn clean verify` → Build failuje zbog uočene nepravilnosti.

TDD (Test-Driven Development) kao Shift-Left stub

Koncept: Pisanje testova pre samog produkcionog koda.

TDD CIKLUS: CRVENO – ZELENO – REFAKTORIŠI

01 CRVENO

Napiši test koji ne prolazi.

02 ZELENO

Napiši minimalan kod da test prođe.

03 REFAKTORIŠI

Unapredi kod bez menjanja funkcionalnosti.

Praktičan primer: TDD postavka za Javalin API

Zadatak

Napraviti REST endpoint /greet koji prima parametar name i vraća JSON poruku.

ZAVISNOSTI U pom.xml

```
<dependency>
    <groupId>io.javalin</groupId>
    <artifactId>javalin</artifactId>
    <version>6.1.3</version>
</dependency>
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter</artifactId>
    <version>5.10.2</version>
    <scope>test</scope>
</dependency>
```

Korak 1: pisanje neuspešnog JUnit testa (Crveno)

JAVA / GREETCONTROLLERTEST

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class GreetControllerTest {
    @Test
    public void testGreetingGeneration() {
        GreetService service = new GreetService();
        // Klasa još ne postoji!
        String result = service.createMessage("DevOps");
        assertEquals("Hello, DevOps!", result);
    }
}
```

Korak 2 & 3: implementacija i Javalin integracija (Zeleno)

IMPLEMENTACIJA SERVISA

```
public class GreetService {
    public String createMessage(String name) {
        return "Hello, " + name + "!";
    }
}
```

JAVALIN APLIKACIJA / KONZOLNO POKRETANJE

```
import io.javalin.Javalin;

public class App {
    public static void main(String[] args) {
        GreetService service = new GreetService();
        Javalin app = Javalin.create().start(7070);
        app.get("/greet", ctx -> {
            String name = ctx.queryParamAsClass(
                "name", String.class).getOrDefault("Guest");
            ctx.result(service.createMessage(name));
        });
    }
}
```

Shift-Right strategija: teorijski koncept

Suština

Testiranje u produkciji sa stvarnim korisnicima i opterećenjem.

Zašto?

Neka ponašanja sistema (mrežna kašnjenja, ponašanje baze pod velikim opterećenjem) nemoguće je verno replicirati u test okruženju.

KLJUČNE TEHNIKE

- **Feature Flags (Zastavice funkcionalnosti)**
- **Canary Deployments (Kanarinac isporuke)**
- **A/B Testiranje**
- **Monitoring i logovanje performansi**

Tehnika: Feature Flags (karakteristične zastavice)

Definicija

Tehnika koja omogućava uključivanje ili isključivanje određenih funkcionalnosti u produkciji bez ponovnog diplomiranja (deploy-a) koda.

Prednost

Razdvajanje tehničkog deployment-a od poslovnog lansiranja (Release).

Rizik

Tehnički dug ako se zastavice ne čiste redovno iz koda.

Praktičan primer: jednostavan Feature Flag u Javi

JAVA / FEATURE TOGGLE SERVICE + KONTROLER

```
public class FeatureToggleService {
    // U realnosti, ovo se povlači iz baze ili servisa poput LaunchDarkly
    public static boolean isNewGreetingAlgoEnabled() {
        return Boolean.parseBoolean(System.getProperty(
            "FEATURE_NEW_GREET", "false"));
    }
}

// Upotreba u kontroleru
app.get("/greet", ctx -> {
    if (FeatureToggleService.isNewGreetingAlgoEnabled()) {
        ctx.result("Welcome to the future, " +
            ctx.queryParam("name") + "!");
    } else {
        ctx.result("Hello, " + ctx.queryParam("name"));
    }
});
```

Tehnika: Canary Deployments i A/B testiranje

Canary Deployment

Isporuka nove verzije aplikacije samo malom procentu servera/korisnika (npr. 5%).

Ako metrike (error rate) skoče, automatski se radi rollback.

A/B testiranje

Statističko poređenje dve verzije (A i B) radi merenja poslovnih performansi (npr. koja verzija donosi više klikova).

Canary prati rizik isporuke; A/B poredi poslovni učinak dve varijante.

Monitoring performansi: Micrometer i Javalin

Uloga monitoringa: Shift-Right testiranje se oslanja na telemetriju. Ako nemamo metrike, slepi smo u produkciji.

PRIMER PRAĆENJA BROJA ZAHTEVA PREKO MICROMETER BIBLIOTEKE

```
import io.micrometer.core.instrument.Counter;
import io.micrometer.prometheus.PrometheusMeterRegistry;

PrometheusMeterRegistry registry =
    new PrometheusMeterRegistry(PrometheusConfig.DEFAULT);
Counter requestCounter = Counter.builder("http_requests_total")
    .description("Ukupan broj HTTP zahteva")
    .register(registry);

app.before(ctx -> requestCounter.increment());
app.get("/metrics", ctx -> ctx.result(registry.scrape()));
```

Kapije kvaliteta (Quality Gates): definicija

Definicija

Automatizovani set matematičkih i metričkih uslova koje kod mora da zadovolji da bi prešao u narednu fazu CI/CD pajplajna.

Uloga

Deluje kao „čuvar kapije“ – sprečava da loš kod stigne do produkcije bez ljudske intervencije.

PRIMER USLOVA

Code Coverage

Pokrivenost koda testovima > 80%.

Critical Vulnerabilities

Nula kritičnih ranjivosti (0 Critical Vulnerabilities).

Pokrivenost koda (Code Coverage) i JaCoCo

Šta je Code Coverage

Metrika koja pokazuje procenat linija koda / grana koje su izvršene tokom pokretanja testova.

JaCoCo (Java Code Coverage)

Standardni Java alat za generisanje izveštaja o pokrivenosti.

VAŽNA NAPOMENA

Visoka pokrivenost ne garantuje kvalitetne testove (možemo imati 100% pokrivenost bez ijednog assert-a), ali niska pokrivenost garantuje netestiran kod.

Konfiguracija JaCoCo Quality Gate-a u Maven-u (1/2)

Postavljanje striktnog limita u pom.xml:

POM.XML / POČETAK KONFIGURACIJE

```
<plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>0.8.11</version>
    <executions>
        <execution>
            <goals><goal>prepare-agent</goal></goals>
        </execution>
        <execution>
            <id>check</id>
            <goals><goal>check</goal></goals>
            <configuration>
                <rules>
                    <rule>
                        <element>BUNDLE</element>
                        <limits>
                            <limit>
                                <counter>LINE</counter>
                                <value>COVEREDRATIO</value>
                                <minimum>0.80</minimum>
                                <!-- 80% pokrivenost -->
```

Konfiguracija JaCoCo Quality Gate-a u Maven-u (2/2)

Nastavak: zatvaranje elemenata konfiguracije i Maven plugina.

POM.XML / ZAVRŠETAK KONFIGURACIJE

```
                        </limit>
                    </limits>
                </rule>
            </rules>
        </configuration>
    </execution>
  </executions>
</plugin>
```

Minimalna vrednost u konfiguraciji: 0.80, odnosno 80% pokrivenosti linija.

Kako izgleda pad na Quality Gate-u?

Scenario

Programer dodaje novu kompleksnu metodu, ali zaboravlja da napiše JUnit test. Pokrivenost cele aplikacije pada na 78%.

IZVRŠAVANJE U CI TERMINALU / BASH

```
$ mvn clean verify
...
[ERROR] Failed to execute goal org.jacoco:jacoco-maven-plugin:check
[ERROR] Coverage check failed: line covered ratio is 0.78,
        but expected minimum is 0.80
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
```

ISHOD **Pajplajn se prekida, kod se ne spaja sa main granom.**

Selekcija i optimizacija testova (Test Suite Minimization)

Problem

Što aplikacija više raste, broj testova se povećava. Izvršavanje svih testova (jediničnih, integracionih, end-to-end) počinje da traje satima.

Zamor testova (Test Fatigue)

Programeri počinju da ignorišu testove ili preskaču CI jer je povratna informacija prespora.

CILJ **Selektovati i izvršiti samo minimalan set testova koji garantuju stabilnost izmenjenog koda.**

Analiza uticaja (Impact Analysis) i graf kontrole toka

KAKO ALGORITAM FUNKCIONIŠE

01 Sistem analizira koje su linije koda izmenjene u Git commit-u.

02 Pomoću grafa kontrole toka mapiraju se klase i metode koje zavise od tih izmena.

03 Izvršavaju se samo oni testovi koji prolaze kroz te modifikovane delove.

Java alati

Razni komercijalni i open-source CI pametni plagini (npr. Infinitest za IDE, ili parcijalno izvršavanje preko Maven naprednih opcija).

Praktični pristupi minimizaciji kroz Maven

Jedinični testovi (brzi)

Pokreću se na svaki commit: mvn test.

Integracioni testovi (spori)

Pokreću se samo pre spajanja (pre-merge) preko maven-failsafe-plugin: mvn verify.

Razdvajanje testova po fazama omogućava brzu proveru pri svakom commit-u i sporiju proveru pre spajanja.

POKRETANJE SPECIFIČNIH TEST GRUPA / MAVEN KLIJENT

```
# Pokretanje testova samo za izmenjeni paket/modul softvera
mvn -pl :moj-modul test
```

Ključni DevOps pojmovi za Continuous Testing

Build Pipeline

Automatizovani niz koraka kroz koje kod prolazi od repozitorijuma do produkcije.

Pre-merge validation

Testiranje koda unutar Pull Request-a pre nego što postane deo glavne grane.

SAST (Static Application Security Testing)

Automatizovana provera ranjivosti u kodu u ranoj fazi.

Metrike u DevOps ekosistemu (DORA metrike)

Kako merimo uspešnost našeg Continuous Testing-a i DevOps praksi? Kroz standardne DORA metrike:

Deployment Frequency

Koliko često isporučujemo kod u produkciju?

Lead Time for Changes

Koliko vremena treba kodu od programerskog stola do produkcije?

Change Failure Rate

Koliki procenat deployment-a rezultuje otkazom u produkciji?

Time to Restore Service

Koliko brzo se oporavljamo od incidenta?

Šta su zapravo DORA metrike?

Poreklo

Skraćenica za DevOps Research and Assessment (DORA) – tim koji je proveo sedmogodišnje istraživanje o tome šta razlikuje visokoefikasne softverske timove od ostalih.

Zlatni standard

Ove četiri metrike su industrijski standard za merenje efikasnosti isporuke softvera i stabilnosti sistema.

Dve strane medalje

Metrike brzine: Deployment Frequency i Lead Time for Changes.
Metrike stabilnosti: Change Failure Rate i Time to Restore Service.

DEVOPS BALANS

Cilj je povećati brzinu bez žrtvovanja stabilnosti.

Metrika 1 – Deployment Frequency (učestalost isporuke)

Definicija

Koliko često tim uspešno pušta kôd u produkciju (ili ga isporučuje krajnjim korisnicima).

Elite timovi

Više puta dnevno (na svaki uspešan merge).

Low-performing timovi

Jednom mesečno ili jednom u nekoliko meseci.

Kako Continuous Testing pomaže: Bez automatizovanih testova, timovi se plaše produkcije i namerno akumuliraju kôd (prave ogromne, rizične verzije). Brzi JUnit i Javalin integracioni testovi u CI daju timu samopouzdanje za česte isporuke.

Metrika 2 – Lead Time for Changes (vreme realizacije izmena)

Definicija

Vreme koje protekne od trenutka kada programer napiše prvu liniju koda (commit) do trenutka kada taj kôd uspešno radi u produkciji.

Glavni usporivači (uska grla)

Tradicionalna faza ručnog QA testiranja koja traje danima ili nedeljama.

Uloga Continuous Testing-a

Pomeranjem testiranja ulevo (Shift-Left) i automatizacijom kroz GitHub Actions, vreme provere se smanjuje sa nekoliko dana na nekoliko minuta.

Kôd prolazi kroz kapije kvaliteta (Quality Gates) u hodu, drastično smanjujući ukupno vreme čekanja.

Metrika 3 – Change Failure Rate (stopa neuspešnih izmena)

Definicija

Procenat deployment-a u produkciju koji rezultuju otkazom sistema, prekidom rada ili zahtevaju hitnu popravku (hotfix/rollback).

FORMULA

CFR = (Broj neuspešnih isporuka / Ukupan broj isporuka) × 100

Praktična veza sa CT-om

Statička analiza (SpotBugs) + Jedinični testovi: eliminišu bazične sintaksne i logičke greške.
JaCoCo Quality Gate: garantuje da nijedna kritična biznis logika (npr. obrada plaćanja) nije otišla u produkciju neproverena.
Rezultat: Stopa neuspeha kod Elite timova pada ispod 5%.

Metrika 4 – Time to Restore Service (vreme oporavka sistema)

Definicija

Koliko je vremena potrebno timu da oporavi aplikaciju kada dođe do nepredviđenog incidenta ili pada u produkciji.

Shift-Right i telemetrija

Kontinuirano testiranje u produkciji preko alata kao što je Micrometer pruža trenutnu telemetriju.

Ako metrika http_requests_total naglo padne ili skoči broj 500 grešaka, automatizovani sistemi odmah alarmiraju tim ili sami pokreću rollback.

Feature Flags

Upotreba Feature Flags (poput primera sa predavanja) omogućava da se neispravna komponenta „ugasi“ jednim klikom u roku od nekoliko sekundi, bez ponovnog postavljanja celog koda.

Antagonizam metrika i kako CT rešava konflikt

Večiti sukob

Razvojni tim (Dev) želi brzinu (gura DORA metrike 1 i 2), dok operativni tim (Ops) želi stabilnost (gura DORA metrike 3 i 4).

Lažni kompromis

Ako samo ubrzate isporuku bez testova, sistem puca (gubi se stabilnost). Ako samo testirate ručno mesecima, gubite korak sa tržištem (gubi se brzina).

ZAJEDNIČKI JEZIK

Continuous Testing je most koji briše ovaj konflikt. Automatizovane kapije kvaliteta u CI/CD omogućavaju maksimalnu brzinu isporuke, dok istovremeno garantuju matematički definisan nivo stabilnosti pre nego što kôd uopšte dotakne server.

Kako Continuous Testing utiče na DORA metrike?

Metrika	Bez Continuous Testing-a	Sa Continuous Testing-om
Deployment Frequency	Retko (strah od greške)	Često (poverenje u pajplajn)
Lead Time for Changes	Dugačko (ručni QA proces)	Kratko (automatske kapije)
Change Failure Rate	Visok (bagovi prolaze skriveni)	Nizak (kapije zaustavljaju greške)

Kreiranje kompletnog CI/CD Pipeline-a (konceptualni dijagram)

FAZA 1	Developer piše kod i pokreće lokalno JUnit & SpotBugs.
FAZA 2	Git Push → Pokreće se GitHub Actions / Jenkins.
FAZA 3	Ekshumacija statičke analize → Kompajliranje → JUnit testovi.
FAZA 4	JaCoCo provera (Quality Gate) → Ako prođe → Build Artifact.
FAZA 5	Deployment na staging/produkciju sa Feature Flags kontrolom.

Zaključak i rezime predavanja

Continuous Testing

Nije luksuz, već neophodnost u brzom razvoju softvera.

Shift-Left

Smanjuje troškove pronalaženja bagova kroz rano testiranje i TDD.

Shift-Right

Omogućava da bezbedno eksperimentišemo i pratimo sisteme na produkciji.

Kapije kvaliteta

Štite integritet sistema bez usporavanja tima.

Kvalitet se gradi i proverava kroz ceo životni ciklus – od prve linije koda do produkcije.